



МЕЖДУНАРОДНА НАУЧНА КОНФЕРЕНЦИЯ „МЕХАТРОНИКА, ЕКО И ЕНЕРГОСПЕСТЯВАЩИ СИСТЕМИ И ТЕХНОЛОГИИ“ - ПЛОВДИВ' 2026

АВТОМАТИЗИРАН ПРОЦЕС ЗА МИГРИРАНЕ ОТ СИСТЕМА ЗА УПРАВЛЕНИЕ НА РЕЛАЦИОННИ БАЗИ ДАННИ КЪМ NOSQL

AUTOMATED PROCESS FOR MIGRATION FROM A RELATIONAL DATABASE MANAGEMENT SYSTEM TO NOSQL

Radoslav Furnadzhiev, Mitko Shopov, Nikolay Kakanakov

1) Technical University of Sofia, Faculty of Electronics and Automation,
Department of Computer Systems and Technologies, Plovdiv, Bulgaria

2) Center of Competence "Smart Mechatronic, Eco-and Energy-Saving Systems and Technologies",
4000 Plovdiv, Bulgaria

Abstract

Driven by the exponential growth of heterogeneous real-time data and the scalability demands of advanced fields such as Intelligent Transportation Systems (ITS), contemporary database management is steadily shifting from relational systems (RDBMS) toward NoSQL data stores. However, manual database migration remains a complex and error-prone process due to the lack of a unified transformation framework. To address this gap, this paper proposes a technology-agnostic, automated four-stage migration pipeline comprising relational metadata extraction, workload-driven schema transformation based on query logs, parallelized bulk data transfer, and a self-correcting verification loop. Each stage is formally conceptualized through detailed pseudocode, integrating schema-mapping, query-mapping, and optional middleware techniques into a unified process. Developed within laboratory C4.1 of the Center of Competence "Smart Mechatronic, Eco-and Energy-Saving Systems and Technologies", the primary contribution of this work lies in synthesizing these distinct methodologies into a single, reproducible framework applicable to document-oriented, column-family, and key-value targets, thereby establishing a solid baseline for future Big Data and experimental evaluations.

Keywords: relational database, NoSQL, automated data migration, schema transformation, MongoDB, big data.

1. INTRODUCTION

Over the last decade, the volume and variety of data produced by web applications, social networks, mobile devices, and IoT systems have grown so rapidly that the relational model, dominant since the 1970s, no longer represents the most appropriate storage paradigm for every application class. Relational database management systems (RDBMS) were designed around a rigid schema and the ACID transactional properties, both of which become difficult to maintain when the workload requires unlimited horizontal scalability, schema flexibility, and high availability across geographically distributed nodes [1,2].

NoSQL data stores emerged as an answer to these limitations. Their less constrained data structures, schema-on-read approach, and design for distribution allow them to manage semi-structured and unstructured data more efficiently than traditional RDBMS [1]. As a consequence, an increasing number of organizations face the practical problem of migrating their existing relational databases to a NoSQL data store, while preserving data integrity and minimizing application-level changes.

Manual migration, however, is a complex and error-prone activity. The lack of a universally accepted, fully automatic



transformation process means that database administrators must combine multiple ad hoc techniques to map relational schemas to document, column-family, or key-value structures and to translate SQL queries into the corresponding NoSQL query languages [3,4]. Numerous partial solutions have been proposed in the literature, ranging from schema-mapping algorithms and middleware adapters to fully automated migration frameworks [2, 4-8], but a unified, technology-agnostic process tailored for engineering laboratories remains an open research topic.

The present paper, prepared within the activities of laboratory C4.1 of the Center of Competence "Smart Mechatronic, Eco-and Energy-Saving Systems and Technologies", addresses this gap by proposing an automated migration process from an RDBMS to a NoSQL database. The main objectives are: (i) to summarise the main approaches reported in the literature; (ii) to identify their strengths and limitations; and (iii) to integrate them into a single, reproducible four-stage process, formalized through pseudocode, that can be used as a baseline for further experimental work in the laboratory. The main limitation of the present study is that it focuses on the methodological side of the migration: a full quantitative experimental evaluation is left for future work.

2. BACKGROUND

2.1. Relational vs. NoSQL data models

An RDBMS stores data in two-dimensional tables connected by primary and foreign keys, enforces strong typing, and provides transactional guarantees through the ACID properties (Atomicity, Consistency, Isolation, Durability). SQL is used both for schema definition and for data manipulation. While very mature and well understood, this paradigm becomes a bottleneck when data sizes reach the petabyte scale and latency-sensitive applications must scale horizontally across many nodes [1,2].

NoSQL is a broad term for a family of non-relational data stores that follow the BASE properties (Basically Available, Soft state, Eventually consistent). They are commonly classified in four categories:

- key-value stores (e.g., Redis, DynamoDB), in which data is stored as pairs of unique keys and arbitrary values;
- document stores (e.g., MongoDB, CouchDB), in which records are semi-structured JSON or BSON documents;
- column-family stores (e.g., Apache Cassandra, HBase), optimized for analytical queries over very large data sets;
- graph databases (e.g., Neo4j, AllegroGraph), designed for highly connected data.

Document-oriented stores, and in particular MongoDB, are the most frequently chosen target for relational-to-NoSQL migrations, mainly because their nested document model can naturally accommodate parent-child relationships through embedding [2,5,6,8].

2.2. Motivation for database migration

The literature consistently identifies the following reasons for migrating from an RDBMS to a NoSQL data store: (i) the inability of relational systems to scale beyond a single high-end server without significant performance degradation; (ii) the need to manage semi-structured and unstructured content (sensor readings, JSON payloads, log files, multimedia metadata); (iii) the demand for continuous availability in cloud-hosted services; and (iv) the lower operating costs offered by horizontally scalable clusters of commodity nodes [1,2,8].

3. RELATED WORK

The problem of automating the relational-to-NoSQL migration has been investigated from several complementary angles, summarised below.

3.1. Schema-mapping approaches

Abo Dabowsa et al. [6] describe a procedure for converting a relational schema into a document-oriented NoSQL cloud database,

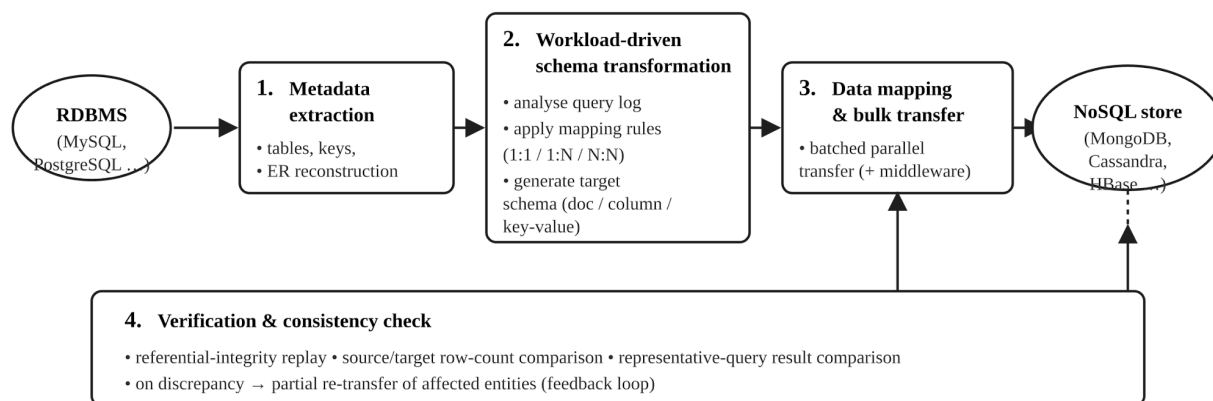


Fig. 1. Overall pipeline of the proposed automated migration process from RDBMS to NoSQL.

with the resulting collections preserving the integrity constraints of the source schema. Ghotiya et al. [8] propose a "mid-model" that captures both the structural characteristics of the source data and the dominant query patterns of the application, so that the destination is organized to favor the most frequent access paths.

An automated algorithm that couples schema reconstruction with subsequent data transformation to minimize manual intervention during the migration from a relational to a NoSQL store is presented in [5].

3.2. Middleware and framework-based approaches

A NoSQLayer is introduced in [7], a framework comprising a migration module that transfers data from MySQL to MongoDB and a mapping module that serves as a persistence layer, translating application requests so that the source code remains essentially unchanged. Experiments reported by the authors show that the overhead introduced by the additional layer becomes negligible as the data volume increases, making the solution particularly attractive for web-scale applications.

3.3. Automatic schema transformation

An automatic SQL-to-NoSQL schema transformation between MySQL and HBase is described in [3]. Tables are converted into column families, and the corresponding queries are rewritten accordingly. The authors report

that, after denormalization, access performance for content-management workloads improved by at least 45%.

3.4. Programming-language-driven transformation

Hasan and Bakar [1] describe a data-transformation pipeline from SQL to MongoDB built on top of the R programming language. The approach exploits R's data-manipulation primitives to convert relational records into corresponding document representations, making it especially convenient for analysts already working in a statistical computing environment.

3.5. End-to-end automated tools

DLoader, an end-to-end approach that transforms data from MySQL into either Cassandra (column-family) or MongoDB (document-oriented), is proposed in [2]. Their experiments suggest that, for many workloads, Cassandra requires less storage space and offers higher query performance after the migration, while MongoDB provides more natural support for nested entities. Yangui et al. [4] add a comparative study of automatic transformation rules from data-warehouse schemas to NoSQL targets, highlighting the trade-offs between target families.

4. PROPOSED AUTOMATED MIGRATION PROCESS



Building on the techniques reviewed in Section 3, we propose a four-stage automated process for migrating from an RDBMS to a NoSQL data store. The process is technology-agnostic on the target side: it specializes in document-oriented, column-family, or key-value stores. Figure 1 summarises the overall pipeline, while Sections 4 formalize each stage in pseudocode.

4.1. Stage 1: Extraction of relational metadata

The first stage queries the source RDBMS's information schema and reconstructs the Entity-Relationship (ER) diagram, including tables, attributes, primary and foreign keys, indexes, and existing constraints. This step is fully automated and serves as an input for all subsequent stages. The output is an internal representation S of the source schema, comparable in spirit to the schema-reconstruction step described by Abo Dabowsa et al. [6]. Algorithm 1 summarises the procedure.

Algorithm 1. Extract relational metadata

```
Input : connection C to source RDBMS
Output : schema S = (T, A, PK, FK, IDX, G)
1. T <- query INFO_SCHEMA.TABLES(C)
2. A <- {}; PK <- {}
3. FK <- {}; IDX <- {}
4. for each table t in T do
5.   A[t] <- columns + types of t
6.   PK[t] <- primary key cols of t
7.   FK[t] <- foreign keys of t
8.   IDX[t] <- secondary indexes
9. end for
10. build ER graph G from (T,PK,FK)
11. return S = (T,A,PK,FK,IDX,G)
```

4.2. Stage 2: Workload-driven schema transformation

Following Ghotiya et al. [8], the destination schema is not derived solely from the source structure; the application's dominant query patterns are also taken into account. The procedure analyses the query log Q to assign a weight $w(e)$ to every edge e of the ER graph, classifies the cardinalities of the source

relationships, and applies three transformation rules:

1) one-to-one relationships are converted into embedded sub-documents (for document targets) or into additional columns within the same column family (for column-family targets);

2) one-to-many relationships are converted into embedded arrays when the cardinality is bounded and into references otherwise;

3) many-to-many relationships are converted either into two-way references or into a dedicated linking collection, depending on whether the typical query traverses the relation in both directions.

When the target is a key-value store, a canonical row-to-pair mapping is used as a fallback, in line with the automated algorithm proposed by Mahmood [5]. Algorithm 2 formalizes the procedure.

Algorithm 2. Workload-driven schema transformation

```
Input : source schema S, query log Q,
       target family F in {DOC,COL,KV}
Output : target schema S'
1. W <- analyse_query_log(Q, S)
2. R <- classify_rel(S) // 1:1,1:N,N:N
3. S' <- {}
4. for each table t in S.T do
5.   e <- new_entity(t)
6.   for each (t,u,card) in R do
7.     if card = '1:1' then
8.       embed(e, u, F)
9.     else if card = '1:N' then
10.      if bounded(u) and W[(t,u)] high
11.        then embed_array(e, u, F)
12.      else reference(e, u, F)
13.     else // N:N
14.      if bidirectional(W,t,u)
15.        then two_way_ref(e, u, F)
16.      else link_collection(e, u, F)
17.   end for
18.   if F = KV then e <- canonical_kv(e)
19.   S'.add(e)
20. end for
21. return S'
```

4.3. Stage 3: Data mapping and bulk transfer

Once the destination schema S' is fixed, the third stage performs the actual data transfer. To handle large volumes, data is transferred in



batches of size B , and transfers are parallelized whenever the target supports concurrent writes. Following the architecture of NoSQLayer [7], a persistence layer P can be inserted between the application and the new database, so that legacy SQL queries are translated on the fly into the corresponding NoSQL operations until the application code is refactored.

Algorithm 3. Data mapping and bulk transfer

Input : S, S' , batch size B , useP flag
Output : populated NoSQL store N

1. open connection N
2. for each e in S' in topo. order do
3. $src \leftarrow origin_tables(e, S)$
4. $offset \leftarrow 0$
5. repeat
6. $rows \leftarrow read_batch(src, offset, B)$
7. $docs \leftarrow map(rows, e, S')$
8. parallel_for d in $docs$ do
9. $N.insert(e.collection, d)$
10. end parallel_for
11. $offset \leftarrow offset + B$
12. until $rows = empty$
13. end for
14. if useP then $deploy_layer(P, S, S')$
15. return N

4.4. Stage 4: Verification and consistency check

The last stage performs three checks: (i) referential integrity is verified by replaying the foreign-key constraints on the destination data; (ii) row counts are compared between source and target; and (iii) a sample of representative queries is executed on both systems, and the results are compared. Any discrepancy triggers a partial re-transfer of the affected entities, closing the feedback loop shown in Figure 1.

Algorithm 4. Verification and consistency check

Input : S, N , sample queries QS
Output : V in $\{OK, REPAIRED, FAIL\}$

1. $iter \leftarrow 0$
2. repeat
3. $marked \leftarrow \{\}$
4. for each e in S' do
5. if $count(e, N) \neq count(orig(e), S)$
6. then $marked.add(e)$
7. end for
8. for each fk in $S.FK$ do
9. if not $replay_fk(fk, N)$ then
10. $marked.add(target_of(fk))$

11. end for
12. for each q in QS do
13. if $exec(q, S) \neq exec(map(q), N)$
14. then $marked.add(entities(q))$
15. end for
16. if $marked = empty$ then return OK
17. $retransfer(marked, S, N)$
18. $iter \leftarrow iter + 1$
19. until $iter > MAX_ITER$
20. return FAIL

5. DISCUSSION

Several recurring observations emerge from comparing the reviewed approaches with the proposed process. First, no single migration recipe fits all NoSQL families: document stores favor denormalization through embedding, column-family stores require careful design of the row key, and key-value stores impose the strongest constraints on the access patterns [1,2,5]. Second, taking the application workload into account is consistently reported as the single most effective factor in obtaining a destination schema that is not just correct but also efficient [2,8]. Third, intermediate middleware layers reduce the cost of application refactoring at the expense of additional runtime overhead, which is acceptable only for large data volumes [4,7].

The proposed process incorporates these three lessons. By keeping the workload analysis as a first-class input of Algorithm 2 and by leaving the use of a middleware layer optional in Algorithm 3, the process can be specialized either for a fully offline migration (with maximum freedom to optimize the destination schema) or for a phased online migration (with minimal disruption to the running application). The verification loop in Algorithm 4 finally guarantees that the migration is rolled back to a consistent state whenever discrepancies are detected.

6. CONCLUSION

This paper has presented a four-stage automated process for migrating data from a relational database management system to a NoSQL data synthesizing techniques previously proposed in isolation in the



literature. Each stage is formalized in pseudocode (Algorithms 1–4) so that the process can be implemented reproducibly. The originality of the contribution lies not in any individual stage but in the integration of schema extraction, workload-driven transformation, data transfer, and post-migration verification into a single, technology-agnostic pipeline that can serve as a methodological baseline for laboratory C4.1 of the Center of Competence.

Future work will focus on three directions: (i) a prototype implementation of Algorithms 1–4, (ii) experimental evaluation on representative datasets, comparing the schemas produced by the process against those obtained by purely structural transformations, and (iii) the extension of the process to graph databases, which are increasingly relevant for highly connected mechatronic and energy data.

Furthermore, the proposed automated migration framework has significant potential for both scientific and practical applications in Intelligent Transportation Systems (ITS). Modern ITS generates huge volumes of heterogeneous real-time data – from GPS telemetry and traffic sensor streams to critical V2X (Vehicle-to-Everything) communications. Efficient management of these dynamic streams requires a smooth transition from traditional relational databases to highly scalable NoSQL repositories. Implementing our four-stage pipeline in such systems would enable automated, error-free transformation of legacy infrastructure, providing the flexibility and low latency needed for Big Data processing in the transportation sector, a promising direction for future applied research.

Acknowledgements

This research was funded by the European Regional Development Fund within the OP “Research, Innovation and Digitalization Programme for Intelligent Transformation 2021–2027”, Project CoC “Smart Mechatronics, Eco- and Energy Saving Systems

and Technologies”, No. BG16RFPR002-1.014-0005.

REFERENCES

- [1] Mahmood, A. A. Automated Algorithm for Data Migration from Relational to NoSQL Databases. *Al-Nahrain Journal for Engineering Sciences*, 21(1), 60-65, 2018. doi: 10.29194/NJES21010060.
- [2] Abo Dabowsa N. I., Maatuk A. M., Elakeili S., Akhtar Ali M. Converting Relational Database to Document-Oriented NoSQL Cloud Database. *IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, pp. 381-386, 2021, doi: 10.1109/MI-STA52233.2021.9464488.
- [3] Ghotiya S., Mandal J., Saravanakumar K. Migration from relational to NoSQL database. *IOP Conference Series: Materials Science and Engineering*, 263(4), 042055, 2017, doi: 10.1088/1757-899X/263/4/042055.
- [4] Hasan F., Bakar M. Data Transformation from SQL to NoSQL MongoDB Based on R Programming Language. *5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Turkey, 2021, pp. 399-403, doi: 10.1109/ISMSIT52890.2021.9604548.
- [5] Lee C.-H., Zheng Y.-L. Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases. *IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*, 2015, doi: 10.1109/ICCE-TW.2015.7216979.
- [6] Yangui R., Nabli A., Gargouri, F. Automatic Transformation of Data Warehouse Schema to NoSQL Data Base: Comparative Study. *Procedia Computer Science*, 96, 255-264, 2016. doi: 10.1016/j.procs.2016.08.138.
- [7] Rocha L., Vale F., Cirilo E., Barbosa D., Mourão F. A Framework for Migrating Relational Datasets to NoSQL. *Procedia Computer Science*, 51, 2593–2602, 2015, doi: 10.1016/j.procs.2015.05.367.
- [8] Rajaram K., Sharma P., Selvakumar S. DLoader: Migration of Data from SQL to NoSQL Databases. In: *Proc. of the Int. Conf. on Cognitive and Intelligent Computing (ICCIC 2021)*, Cognitive Science and Technology, Springer, Singapore, 2023, pp. 211–222, doi: 10.1007/978-981-19-2358-6_19.